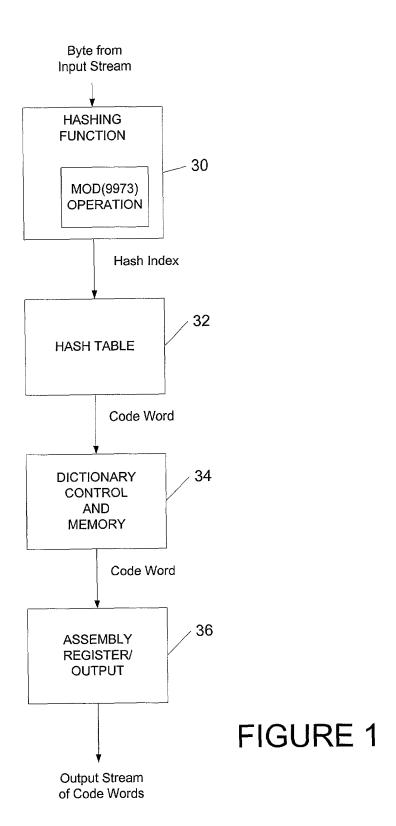
Inventors: Joseph H. End III
Title: "CIRCUIT AND METHOD FOR HIGHSPEED EXECUTION OF MODULO DIVISION"
Sheet 1/12



Inventors: Joseph H. End III
Title: "CIRCUIT AND METHOD FOR HIGHSPEED EXECUTION OF MODULO DIVISION"
Sheet 2/12

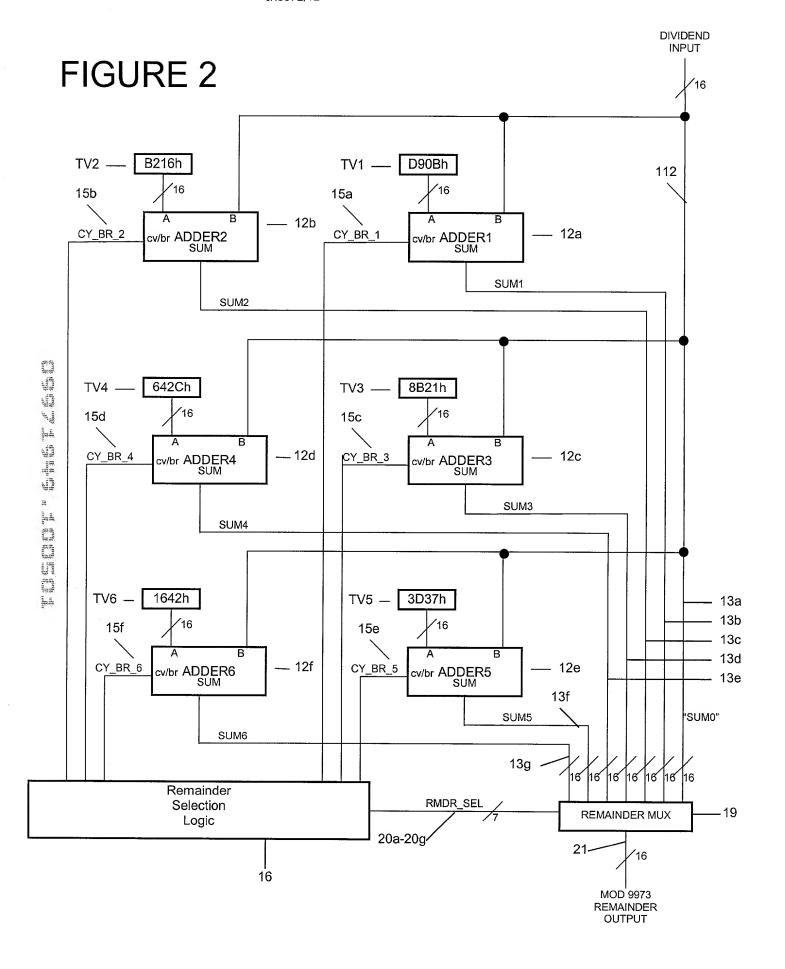
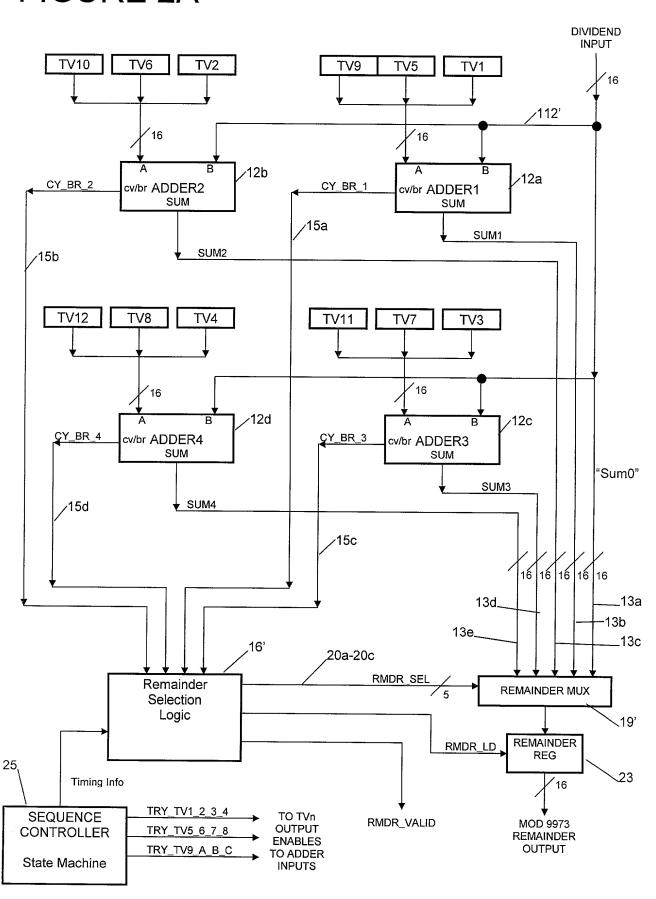


FIGURE 2A

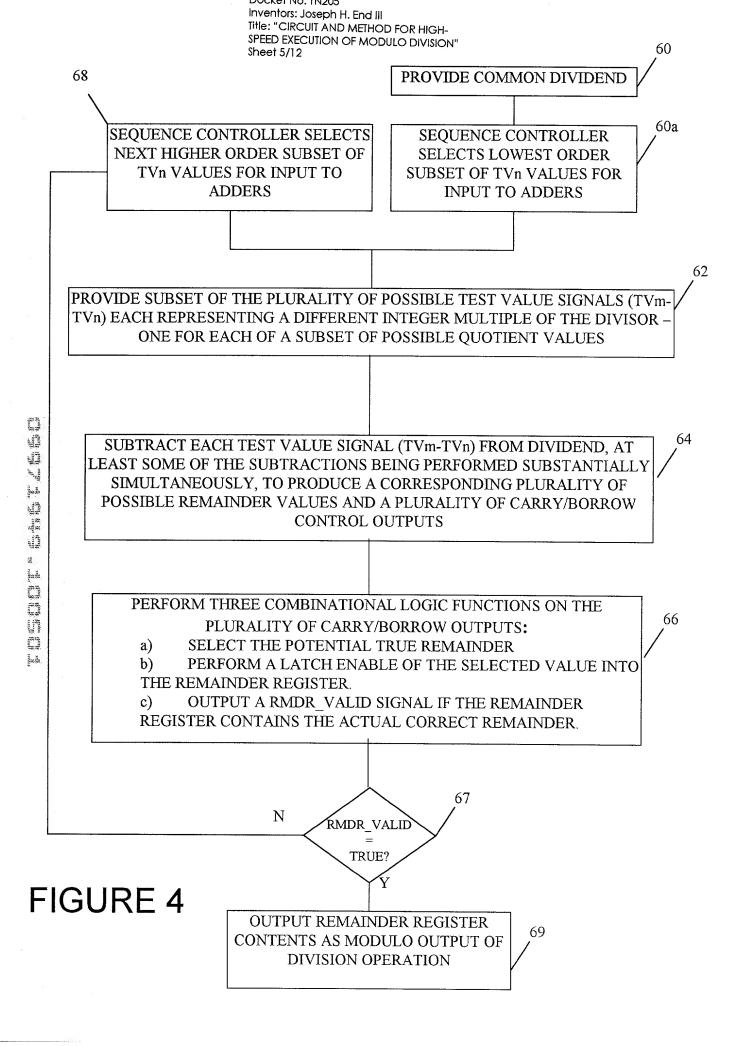


Sheet 4/12 50 **PROVIDE** COMMON DIVIDEND PROVIDE PLURALITY OF TEST VALUE SIGNALS (TV0-TVN) EACH 52 REPRESENTING A DIFFERENT INTEGER MULTIPLE OF THE DIVISOR - ONE FOR EACH POSSIBLE QUOTIENT VALUE SUBTRACT EACH TEST VALUE SIGNAL (TV0-TVN) FROM DIVIDEND, AT LEAST SOME OF THE SUBTRACTIONS BEING 54 PERFORMED SUBSTANTIALLY SIMULTANEOUSLY, TO PRODUCE A CORRESPONDING PLURALITY OF POSSIBLE REMAINDER VALUES AND A PLURALITY OF CARRY/BORROW **CONTROL OUTPUTS** PERFORM COMBINATORIAL LOGIC 56 FUNCTION ON THE COLLECTIVE CARRY/ BORROW OUTPUTS TO SELECT THE CORRECT REMAINDER 58 **OUTPUT SELECTED REMAINDER AS** MODULO OUTPUT OF DIVISION **OPERATION**

Inventors: Joseph H. End III

Title: "CIRCUIT AND METHOD FOR HIGH-SPEED EXECUTION OF MODULO DIVISION"

FIGURE 3



Inventors: Joseph H. End III
Title: "CIRCUIT AND METHOD FOR HIGHSPEED EXECUTION OF MODULO DIVISION"
Sheet 6/12

All values shown are decimal except "K" is 2^10 (1024 decimal) multiplier.

	2K 2K 2K 2K 2K	2K 2K 2K 2K	2K 2	2K 2K 2K 2K	2K 2K 2K 2K 2K	2K 2K 2K 2K 2	K 2K 2K
			64	64K (16 bits)			1 1
Range Width	9973	9973	9973	9973	8973	8266	5698
DIVIDEND	0000	973	19945 19946 29918 29919		39892	49864 49865 59837	ດັ
MOD Remainder Output Quotient	0=0 0=0	0 9972 0 Q=1	0 9972 0 Q=2	Q=3	0 9972 0 Q=4	ğ	0 5697 Q=6
Cv/Br. "Adder 0"	1 1 1 1 1 1 1 1	: : : : :	1				_
	0000000000	111111111	11111111	111111	11111111111	11111111	
	0000000000	00000	11111	1111111111	111111111	111111111111111111111111111111111111111	
	0000000000	000000000	000000	11111	111111111	1111111	1111
	0000000000	000000000	000000000	00000000	1111	11111111	1111
Cy/Br, Adder 5, "15e" Cy/Br, Adder 6, "15f"	000000000000	000000000000000000000000000000000000000	00000000	00000000	00	1111111111	
	7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7				, , , , , , , , , , , , , , , , , , ,		
	00000000000					00000000	0000
	0000000000	00000000	11111111111	00000000			
	0000000000	00000000	000000000	111111111	0000000000	0	0000
	00000000000	00000000	000000000	00000000	111111111	0000000	0000
	00000000000	00000000	00000	00000000	000000000	111111	0000
Sel Sum 6	00000000000	00000000000	00000000000	0000000	000000000	00000000	1111

FIGURE 5

Inventors: Joseph H. End III
Title: "CIRCUIT AND METHOD FOR HIGHSPEED EXECUTION OF MODULO DIVISION"
Sheet 7/12

FIGURE 6A

```
--Selects which of the adder output values is the Remainder Output.
    Signal RMDR_SEL : std_logic_vector( 6 downto 0); -- 7 bit vector of signals used
         -- to select the true remainder from the plurality of remainders.
         -- The vector is "one hot" encoded.
    Signal CY_BR_VEC: std_logic_vector(5 downto 0); -- Vector of CY_BR1...6 bits.
         -- composed of the plurality of carry/borrow outputs of the adders.
    Signal QUOTIENT_VAL: std_logic_vector(2 downto 0); -- Numerical value of quotient.
-- Adder 0 was reduced out of the design and thus eliminated, being replaced with
       a straight-through bus, leaving the other 6 adders physically implemented.
-- Create a vector from the adder CY_BR outputs so they can be referenced as a group
-- in the CASE statement in the RMDR_SEL_LOGIC process, below.
  CY_BR_VEC(0) <= CY_BR1; -- CY_BR output of adder 1 is bit 0 of CY_BR_VEC.
  CY_BR_VEC(1) <= CY_BR2; -- CY_BR output of adder 2 is bit 1 of CY_BR_VEC.
    CY_BR_VEC(2) <= CY_BR3; -- CY_BR output of adder 3 is bit 2 of CY_BR_VEC.
  CY_BR_VEC(3) <= CY_BR4; -- CY_BR output of adder 4 is bit 3 of CY_BR_VEC.
  CY_BR_VEC(4) <= CY_BR5; -- CY_BR output of adder 5 is bit 4 of CY_BR_VEC.
  CY_BR_VEC(5) <= CY_BR6; -- CY_BR output of adder 6 is bit 5 of CY_BR_VEC.
  -- Remainder Select control output to control Remainder Mux, item # 19.
RMDR_SEL_LOGIC : process (CY_BR_VEC)
---RMDR_SEL output signal names and values.
                                                  "One-hot" encoded.
Constant ZERO : std_logic_vector(6 downto 0) := "0000001"; -- Select "adder 0" output.
Comstant ONE : std_logic_vector(6 downto 0) := "0000010"; -- Select adder 1 output.
Constant TWO : std_logic_vector(6 downto 0) := "0000100"; -- Select adder 2 output.
Constant THREE: std_logic_vector(6 downto 0) := "0001000"; -- Select adder 3 output.
Constant FOUR : std_logic_vector(6 downto 0) := "00100000"; -- Select adder 4 output. Constant FIVE : std_logic_vector(6 downto 0) := "01000000"; -- Select adder 5 output.
Constant SIX : std_logic_vector(6 downto 0) := "1000000"; -- Select adder 6 output.
-- CY_BR_VEC names and values which match associated numerical quotient values.
Constant QUOT_0 : std_logic_vector(5 downto 0) := "000000"; -- Quotient of dividend is 0. Constant QUOT_1 : std_logic_vector(5 downto 0) := "000001"; -- Quotient of dividend is 1.
Constant QUOT_2 : std_logic_vector(5 downto 0) := "000011";
                                                                   -- Quotient of dividend is 2.
Constant QUOT_3 : std_logic_vector(5 downto 0) := "000111";
                                                                   -- Quotient of dividend is 3.
Constant QUOT_4 : std_logic_vector(5 downto 0) := "001111";
                                                                  -- Quotient of dividend is 4.
Constant QUOT_5 : std_logic_vector(5 downto 0) := "011111"; -- Quotient of dividend is 5. Constant QUOT_6 : std_logic_vector(5 downto 0) := "111111"; -- Quotient of dividend is 6.
```

Inventors: Joseph H. End III
Title: "CIRCUIT AND METHOD FOR HIGHSPEED EXECUTION OF MODULO DIVISION"
Sheet 8/12

110 111200

FIGURE 6B

```
-- Perform selection function.
-- Note: Interpretation of the CASE statement. Example: "WHEN QUOT_0 =>" means, "When the
-- value of CY_BR_VEC equals the value assigned to the constant, QUOT_0, execute the
-- following statements until the next "WHEN..." statement". Then exit the CASE.
Case CY_BR_VEC is
  When QUOT 0 =>
                 -- Quotient of dividend is 0. "Adder 0" has correct mod output value.
     RMDR SEL
              <= ZERO;
                            -- Select remainder 0.
     QUOTIENT VAL <= "000" -- Quotient value is 0 (binary).
                  -- Quotient of dividend is 1. Adder 1 has correct mod output value.
  When QUOT 1 =>
     RMDR SEL
                 <= ONE;
                                 -- Select remainder 1.
     QUOTIENT VAL <= "001" -- Quotient value is 1 (binary).
 When QUOT_2 => -- Quotient of dividend is 2. Adder 2 has correct mod output value.
     RMDR_SEL <= TWO; -- Select remainder 2.
     QUOTIENT_VAL <= "010" -- Quotient value is 2 (binary).
 When QUOT_3 => -- Quotient of dividend is 3. Adder 3 has correct mod output value.
    RMDR SEL <= THREE;
                           -- Select remainder 3.
     QUOTIENT_VAL <= "011" -- Quotient value is 3 (binary).
 When QUOT_4 =>
                 -- Quotient of dividend is 4. Adder 4 has correct mod output value.
     RMDR SĒL
              <= FOUR;
                                 -- Select remainder 4.
     QUOTIENT_VAL <= "100" -- Quotient value is 4 (binary).
 When QUOT_5 => -- Quotient of dividend is 5. Adder 5 has correct mod output value.
     RMDR SEL <= FIVE;
                                 -- Select remainder 5.
     QUOTIENT_VAL <= "101" -- Quotient value is 5 (binary).
 When QUOT_6 => -- Quotient of dividend is 6. Adder 6 has correct mod output value.
     RMDR SEL <= SIX;
                          -- Select remainder 6.
     QUOTIENT VAL <= "110" -- Quotient value is 6 (binary).
```

End case;

Inventors: Joseph H. End III
Title: "CIRCUIT AND METHOD FOR HIGHSPEED EXECUTION OF MODULO DIVISION"
Sheet 9/12

0,,,0 114200

FIGURE 7A

Note: Interpretation of VHDL:

a) CASE statement. Example:

Case SEQ_STATE is

WHEN CYCLE_1 then

"WHEN CYCLE_1" means, "When the value of SEQ_STATE equals the value assigned to the constant, CYCLE_1, execute the statements following the WHEN until the next "WHEN..." statement". Then exit the CASE.

b) The symbol "<=" is interpreted as: "is assigned the value of...".

```
********* BEGINNING OF "VHDL" DESCRIPTION *******************
--Signal declarations and definitions.
   Signal CY_BR VEC: std_logic vector(3 downto 0); -- 4 bit vector of CY BR signals.
        -- composed of the plurality of carry/borrow outputs of the adders.
   Signal QUOTIENT_VAL: std logic_vector(3 downto 0); -- 4 bit numeric value of quotient.
   Signal SEQ_STATE : std_logic_vector(1 downto 0); -- Sequence Controller state vector. Signal SEQ_STATE_N: std_logic_vector(1 downto 0); -- Seq Controller next state vector.
-- Output signals of Remainder Selection Logic, 16
 Signal RMDR_LD: std_logic; -- Remainder Reg load-enable signal.
 Signal RMDR_SEL: std_logic_vector( 5 downto 0); -- 6 bit vector of signals used
        -- to select the true remainder from the plurality of remainders.
        -- The vector is "one hot" encoded.
 Signal RMDR_VALID: std_logic; -- Tags the contents of the Remainder Reg as valid.
- Signals to select subsets of TVn values to apply to adder inputs "A".
 Signal SEL_SET_1: std_logic; -- Signal to apply TV1, TV2, TV3 and TV4.
 Signal SEL_SET_2 : std_logic; -- Signal to apply TV5, TV6, TV7 and TV8.
   Signal SEL_SET_3 : std logic; -- Signal to apply TV9, TV10, TV11 and TV12.
Adder 0 was reduced out of the design and thus eliminated, being replaced with
a straight-through bus, leaving the other 6 adders physically implemented.
- T Create a vector from the adder CY_BR outputs so they can be referenced as a group
in the CASE statement in the RMDR_SEL_LOGIC process, below.
 CY_BR_VEC(0) <= CY_BR1; -- CY_BR output of adder 1 is bit 0 of CY_BR_VEC.
CY_BR_VEC(1) <= CY_BR2; -- CY_BR output of adder 2 is bit 1 of CY_BR_VEC.
 CY BR VEC(2) <= CY BR3; -- CY BR output of adder 3 is bit 2 of CY BR VEC.
  CY BR VEC(3) <= CY_BR4; -- CY BR output of adder 4 is bit 3 of CY_BR_VEC.
```

Inventors: Joseph H. End III
Title: "CIRCUIT AND METHOD FOR HIGHSPEED EXECUTION OF MODULO DIVISION"
Sheet 10/12

FIGURE 7B

```
Behavior of Sequence Controller State Machine, 25 ***********
SEQUENCE_CONTROLLER : process (RMDR_VALID, RESET) -- RMDR_VALID and RESET are input
                                                                                                                                       -- signals.
-- The RMDR VALID signal is generated in the RMDR_SEL_LOGIC process, below.
-- Sequence Controller state names and values. Values are arbitrary.
Constant CYCLE_1 : std_logic_vector(1 downto 0) := "01"; -- Controller CYCLE 1 state.
Constant CYCLE_2 : std_logic_vector(1 downto 0) := "10"; -- Controller CYCLE_2 state. Constant CYCLE_3 : std_logic_vector(1 downto 0) := "11"; -- Controller CYCLE_3 state.
-- Sequencer behavior
         If RESET = 1' then
                SEQ_STATE <= CYCLE_1; -- Reset to CYCLE 1 state.</pre>
        else -- Normal sequence controller operation.
                Case SEQ STATE is
                        WHEN CYCLE 1 then
                                If RMDR_VALID = '1' then -- The true remainder is in this set.
                                        SEQ STATE N <= CYCLE_1; -- Stay in CYCLE_1 state.</pre>
                                                                                 -- True remainder is not \overline{in} this set.
                                        SEQ_STATE N <= CYCLE_2; -- Continue on to CYCLE_2 state.
                                End if;
                        WHEN CYCLE 2 then
    The state of the s
                                If RMDR VALID = '1' then -- The true remainder is in this set.
    J.
                                        SEQ STATE N <= CYCLE 1; -- Go back to CYCLE_1 state.
                                                                                  -- True remainder is not in this set.
                                        SEQ_STATE_N <= CYCLE_3; -- Continue on to CYCLE_3 state.
                               End if;
    Will Will
                        WHEN CYCLE 3 then -- True remainder MUST be in this set if not found so far.
                                        SEQ_STATE_N <= CYCLE_1; -- Return to CYCLE_1 state.</pre>
               End Case;
End SEQUENCE_CONTROLLER process;
```

1

Inventors: Joseph H. End III
Title: "CIRCUIT AND METHOD FOR HIGHSPEED EXECUTION OF MODULO DIVISION"
Sheet 11/12

FIGURE 7C

```
Behavior of Remainder Selection Logic, 16 **********
-- Remainder Select control output to control Remainder Mux, item # 19.
RMDR_SEL_LOGIC : process (SEQ_STATE,CY_BR_VEC) -- SEQ_STATE and CY_BR_VEC are input
                                             -- signals.
Note that RMDR VALID signal output here is an input to the Sequencer Controller, above.
-- RMDR SEL output signal names and values. "One-hot" encoded.
Constant NONE : std_logic_vector(4 downto 0) := "00000"; -- Don't select any outputs.
Constant ZERO : std_logic_vector(4 downto 0) := "00001"; -- Select "adder 0" output.
Constant ONE : std_logic_vector(4 downto 0) := "00010"; -- Select adder 1 output.
Constant TWO : std logic vector(4 downto 0) := "00100"; -- Select adder 2 output.
Constant THREE: std_logic_vector(4 downto 0) := "01000"; -- Select adder 3 output.
Constant FOUR : std logic vector(4 downto 0) := "10000"; -- Select adder 4 output.
-- CY_BR_VEC names and values which match associated numerical quotient values.
Constant QUOT_A : std logic_vector(3 downto 0) := "0000"; -- Quotient value is 0,4,8,12.
Constant QUOT_B : std_logic_vector(3 downto 0) := "0001"; -- Quotient value is 1,5 or 9.
Constant QUOT_C : std_logic_vector(3 downto 0) := "0011"; -- Quotient value is 2,6 or 10.
Constant QUOT_D : std_logic_vector(3 downto 0) := "0111"; -- Quotient value is 3,7 or 11.
Constant QUOT E : std logic vector(3 downto 0) := "1111"; -- Quotient value is unknown.
Case SEQ STATE is
                                 -- Each value of SEQ_STATE selects a different set of
                                 -- combinational logic to be performed.
 When CYCLE_1 then -- Perform sequencer cycle 1 logic function.
     SEL SET_1 <= '1'; -- Apply TV1, TV2, TV3 and TV4 to adders.
 7
     Case CY_BR_VEC is -- Look at the CY BR adder outputs.
 200
                        -- Quotient of d\overline{i}vidend is 0. This is the only state in which
        When QUOT A =>
 T.
                         -- the "adder 0" output is considered.
                        <= ZERO; -- ."Adder 0" has correct mod output value.
           RMDR SEL
           QUOTIENT VAL <= "0000"; -- Quotient value is 0 (binary).
 1
                       <= `1';
           RMDR VALID
                                   -- Remainder is valid.
                                  -- Load valid remainder into remainder reg, 23.
                        <= \1';
           RMDR LD
 jak.
                        -- Quotient of dividend is 1.
        When QUOT B =>
 RMDR SEL
                        <= ONE; -- Adder 1 has correct remainder output value.</pre>
           QUOTIENT_VAL <= "0001"; -- Quotient value is 1 (binary).
 No.
           RMDR VALID
                       <= \1';
                                    -- Remainder is valid.
                                  -- Load valid remainder into remainder reg, 23.
                        <= \1';
           RMDR LD
        When QUOT C =>
                        -- Quotient of dividend is 2.
           RMDR VALID <= '1';
                                   -- Remainder is valid.
                                  -- Load valid remainder into remainder reg, 23.
                        <= '1';
           RMDR LD
                       -- Quotient of dividend is 3.
        When QUOT D =>
           RMDR SEL
                        <= THREE; -- Adder 3 has correct remainder output value.
           QUOTIENT VAL <= "0011";
                                  -- Quotient value is 3 (binary).
           RMDR VALID
                       <= \1';
                                   -- Remainder is valid.
           RMDR LD
                        <= `1';
                                   -- Load valid remainder into remainder reg, 23.
        When QUOT E =>
                       -- Quotient of dividend might be 4.
           RMDR SEL
                        <= FOUR; -- Adder 4 may have correct remainder output value.
           RMDR VALID <= '0';</pre>
                                   -- Remainder is unknown. Sequencer must continue.
           RMDR LD
                       <= `1';
                                   -- Load remainder into remainder reg, 23. It might
                                   -- be valid. Must be tested next cycle.
     End case;
```

Inventors: Joseph H. End III
Title: "CIRCUIT AND METHOD FOR HIGHSPEED EXECUTION OF MODULO DIVISION"
Sheet 12/12

FIGURE 7D

```
When CYCLE_2 then -- Perform sequencer cycle_2 logic function.
      SEL_SET_2 <= '1'; -- Apply TV5, TV6, TV7 and TV8 to adders.
      Case CY_BR VEC is -- Look at the CY_BR adder outputs.
         When QUOT A => -- Quotient of dividend is 4. This confirms the possibility.
            RMDR SEL
                         <= NONE; -- .All adder outputs are incorrect.</pre>
            QUOTIENT VAL <= "0100"; -- Quotient value is 4 (binary).
            RMDR VALID <= '1';
                                      -- Remainder in Remainder Reg, 23, is valid.
                          <= '0';
            RMDR LD
                                      -- Don't load. Hold valid remainder previously
                                      -- loaded into remainder reg, 23, in CYCLE_1.
         When QUOT B => -- Quotient of dividend is 5.
            RMDR SEL
                         \neq ONE;
                                    -- Adder 1 has correct remainder output value.
            QUOTIENT_VAL <= "0101"; -- Quotient value is 5 (binary).
            RMDR VALID
                        <= `1';
                                      -- Remainder is valid.
            RMDR LD
                          <= '1';
                                    -- Load valid remainder into remainder reg, 23.
         When QUOT C => -- Quotient of dividend is 6.
                         <= TWO; -- Adder 2 has correct remainder output value.</pre>
            RMDR SEL
            QUOTIENT VAL <= "0110"; -- Quotient value is 6 (binary).
                        <= '1';
            RMDR VALID
                                    -- Remainder is valid.
                         <= '1'; -- Load valid remainder into remainder reg, 23.
            RMDR LD
         When QUOT D \Rightarrow -- Quotient of dividend is 7.
                         <= THREE; -- Adder 3 has correct remainder output value.
            RMDR SEL
            QUOTIENT VAL <= "0111"; -- Quotient value is 7 (binary).
            RMDR VALID <= '1';
                                      -- Remainder is valid.
 177
                         <= `1';
            RMDR LD
                                      -- Load valid remainder into remainder reg, 23.
  Section 1
         When QUOT E =>
                         -- Quotient of dividend might be 8.
            RMDR SEL
                         <= FOUR; -- Adder 4 may have correct remainder output value.</pre>
  4
            RMDR_VALID <= '0';</pre>
                                      -- Remainder is unknown. Sequencer must continue.
 1 in.
                         <= `1';
            RMDR LD
                                      -- Load remainder into remainder reg, 23. It might
  100
                                      -- be valid. Must be tested next cycle.
      End case;
  When CYCLE_3 then -- Perform sequencer cycle 3 logic function.
      SEL SET_3 <= '1'; -- Apply TV9, TV10, TV11 and TV12 to adders.
      Case CY_BR VEC is -- Look at the CY BR adder outputs.
 5-2
         When QUOT A =>
                         -- Quotient of dividend is 8. This confirms the possibility.
            RMDR SEL
                                     -- .All adder outputs are incorrect.
-- Quotient value is 8 (binary).
                         <= NONE;
 Second
Second
            QUOTIENT VAL <= "1000";
            RMDR VALID <= '1';
  The state of
                                      -- Remainder in Remainder Reg, 23, is valid.
            RMDR LD
                         <= `0';
                                      -- Don't load. Hold valid remainder previously
                                      -- loaded into remainder reg, 23, in CYCLE_1.
 When QUOT B =>
                         -- Quotient of dividend is 9.
            RMDR SEL
                         <= ONE;
                                    -- Adder 1 has correct remainder output value.
            QUOTIENT VAL <= "1001";
                                      -- Quotient value is 9 (binary).
                        <= `1';
            RMDR VALID
                                      -- Remainder is valid.
            RMDR LD
                         <= '1';
                                     -- Load valid remainder into remainder reg, 23.
                         -- Quotient of dividend is 10.
         When QUOT C =>
            RMDR SEL
                         <= TWO; -- Adder 2 has correct remainder output value.
            QUOTIENT VAL <= "1010"; -- Quotient value is 10 (binary).
            RMDR VALID <= '1';
                                     -- Remainder is valid.
                                    -- Load valid remainder into remainder reg, 23.
            RMDR LD
                         <= `1';
         When QUOT D =>
                         -- Quotient of dividend is 11.
            RMDR SEL
                         <= THREE; -- Adder 3 has correct remainder output value.
            QUOTIENT VAL <= "1011";
                                     -- Quotient value is 11 (binary).
            RMDR_VALID
                        <= `1';
                                     -- Remainder is valid.
            RMDR LD
                         <= `1';
                                     -- Load valid remainder into remainder reg, 23.
                         -- Quotient of dividend is 12.
         When QUOT E =>
            RMDR SEL
                         <= FOUR; -- Adder 4 has correct remainder output value.
            QUOTIENT_VAL <= "1100"; -- Quotient value is 12 (binary).
            RMDR_VALID <= '1'; -- Remainder is valid.

RMDR_LD <= '1'; -- Load valid remainder into remainder reg, 23.
      End case;
End case;
```